Accuractely Computing $\log(1 - \exp(-|a|))$

Martin Mächler
ETH Zurich
April 2012 (LaTeX'ed April 28, 2012)

Abstract

In this note, we explain how $f(a) = \log(1 - e^{-a}) = \log(1 - \exp(-a))$ can be computed accurately, in a simple and optimal manner, building on the two related auxiliary functions `log1p(x)` $(= \log(1 + x))$ and `expm1(x)` $(= \exp(x) - 1 = e^x - 1)$. The cutoff, a_0, in use in R since 2004, is shown to be optimal both theoretically and empirically, using **Rmpfr** high precision arithmetic.

Keywords: Accuracy, Cancellation Error, R, MPFR, Rmpfr.

1. Introduction: Not log() nor exp(), but log1p() and expm1()

In applied mathematics, it has been known for a very long time that direct computation of $\log(1 + x)$ suffers from severe cancellation (in "$1 + x$") whenever $|x| \ll 1$, and for that reason, we have provided `log1p(x)` in R, since R version 1.0.0 (released, Feb. 29, 2000). Similarly, `log1p()` has been provided by C math libraries and has become part of C language standards around the same time, see, for example, IEEE and Open Group (2004).

Analogously, since R 1.5.0 (April 2002), the function `expm1(x)` computes $\exp(x) - 1 = e^x - 1$ accurately also for $|x| \ll 1$, where $e^x \approx 1$ is (partially) cancelled by "$-\,1$".

In both cases, a simple solution for small $|x|$ is to use a few terms of the Taylor series, as

$$\mathrm{log1p}(x) = \log(1 + x) = x - x^2/2 + x^3/3 - + \dots, \quad \text{for} \quad |x| < 1, \tag{1}$$
$$\mathrm{expm1}(x) = \exp(x) - 1 = x + x^2/2! + x^3/3! + \dots, \quad \text{for} \quad |x| < 1, \tag{2}$$

and $n!$ denotes the factorial.

We have found, however, that in some situations, the use of `log1p()` and `expm1()` may not be sufficient to prevent loss of numerical accuracy. The topic of this note is to analyze the important case of computing $\log(1 - e^x) = \log(1 - \exp(x))$ for $x < 0$, computations needed in accurate computations of the beta, gamma, Weibull and logistic distributions, and even for the (inverse logit) link function in logistic regression, see, for example,DiDonato and Morris (1992)[1], and further references mentioned in R's `?pgamma` and `?pbeta` help pages.

2. log1p() and expm1() for log(1 - exp(x))

Contrary to what one would expec, for computing $\log(1 - e^x) = \log(1 - \exp(x))$ for $x < 0$,

[1]In the Fortran source, file "`708`", also available as `http://www.netlib.org/toms/708`, the function ALNREL() computes log1p() and REXP() computes expm1().

neither

$$\log(1 - \exp(x)) = \log(-\text{expm1}(x)), \quad \text{nor} \tag{3}$$

$$\log(1 - \exp(x)) = \text{log1p}(-\exp(x)), \tag{4}$$

are uniformly sufficient for numerical evaluation. In (4), when x approaches 0, $\exp(x)$ approaches 1 and loses accuracy. In (3), when x is large, $\text{expm1}(x)$ approaches -1 and similarly loses accuracy. Because of this, we will propose to use a function `log1mexp(x)` which uses either `expm1` (3) or log1p (4), where appropriate. Already in R 1.9.0 (R Development Core Team (2004)), we have defined the macro `R_D_LExp(x)` to provide these two cases automatically[2].

To investigate the accuracy losses empirically, we make use of the R package **Rmpfr** for arbitrarily accurate numerical computation, and use the following simple functions:

```
> library(Rmpfr)
> t3.l1e <- function(a)
  {
      c(def   = log(1 - exp(-a)),
        expm1 = log( -expm1(-a)),
        log1p = log1p(-exp(-a)))
  }
> ##' The relative Error of log1mexp computations:
> relE.l1e <- function(a, precBits = 1024) {
      stopifnot(is.numeric(a), length(a) == 1, precBits > 50)
      da <- t3.l1e(a)  ## double precision
      a. <- mpfr(a, precBits=precBits)
      ## high precision *and* using the correct case:
      mMa <- if(a <= log(2)) log(-expm1(-a.)) else log1p(-exp(-a.))
      structure(as.numeric(1 - da/mMa), names = names(da))
  }
```

where the last one, `relE.l1e()` computes the relative error of three different ways to compute $\log(1 - \exp(-a))$ for positive a (instead of computing $\log(1 - \exp(x))$ for negative x).

```
> a.s <- 2^seq(-55, 10, length = 256)
> ra.s <- t(sapply(a.s, relE.l1e))
> cbind(a.s, ra.s) # comparison of the three approaches

                a.s          def         expm1         log1p
  [1,] 2.775558e-17         -Inf -7.975523e-17          -Inf
  [2,] 3.311947e-17         -Inf -4.907551e-17          -Inf
  [3,] 3.951996e-17         -Inf -7.870434e-17          -Inf
  [4,] 4.715738e-17         -Inf -4.599819e-17          -Inf
  [5,] 5.627076e-17 1.816201e-02 -7.394731e-17  1.816201e-02
  [6,] 6.714535e-17 1.350365e-02 -4.492137e-17  1.350365e-02
  [7,] 8.012150e-17 8.800880e-03 -1.294478e-17  8.800880e-03
 .......
 .......
[251,] 4.232864e+02 1.000000e+00  1.000000e+00 -3.315145e-17
[252,] 5.050884e+02 1.000000e+00  1.000000e+00  2.926091e-17
[253,] 6.026991e+02 1.000000e+00  1.000000e+00  1.737660e-17
[254,] 7.191735e+02 1.000000e+00  1.000000e+00 -4.726899e-12
[255,] 8.581571e+02 1.000000e+00  1.000000e+00  1.000000e+00
[256,] 1.024000e+03 1.000000e+00  1.000000e+00  1.000000e+00
```

This is revealing: Neither method, log1p or expm1, is uniformly good enough. Note that for large a, the relative errors evaluate to 1. This is because all three double precision

[2]look for "log(1-exp(x))" in `http://svn.r-project.org/R/branches/R-1-9-patches/src/nmath/dpq.h`

methods give 0, *and* that is the best approximation in double precision (but not in higher `mpfr` precision), hence no problem at all, and we can restrict ourselves to smaller a (smaller than about 710, here).

What about really small a's?

```
> t3.l1e(1e-20)

     def    expm1    log1p
    -Inf -46.0517     -Inf

> as.numeric(t3.l1e(mpfr(1e-20, 256)))

[1] -46.0517 -46.0517 -46.0517
```

so, indeed, the `expm1` method is absolutely needed here.

```
> matplot(a.s, abs(ra.s), type = "l", log = "xy",
          col=cc, lty=lt, lwd=ll, xlab = "a", ylab = "", axes=FALSE)
> legend("top", leg, col=cc, lty=lt, lwd=ll, bty="n")
> draw.machEps <- function(alpha.f = 1/3, col = adjustcolor("black", alpha.f)) {
      abline(h = .Machine$double.eps, col=col, lty=3)
      axis(4, at=.Machine$double.eps, label=quote(epsilon[c]), las=1, col.axis=col)
  }
> eaxis(1); eaxis(2); draw.machEps(0.4)
```

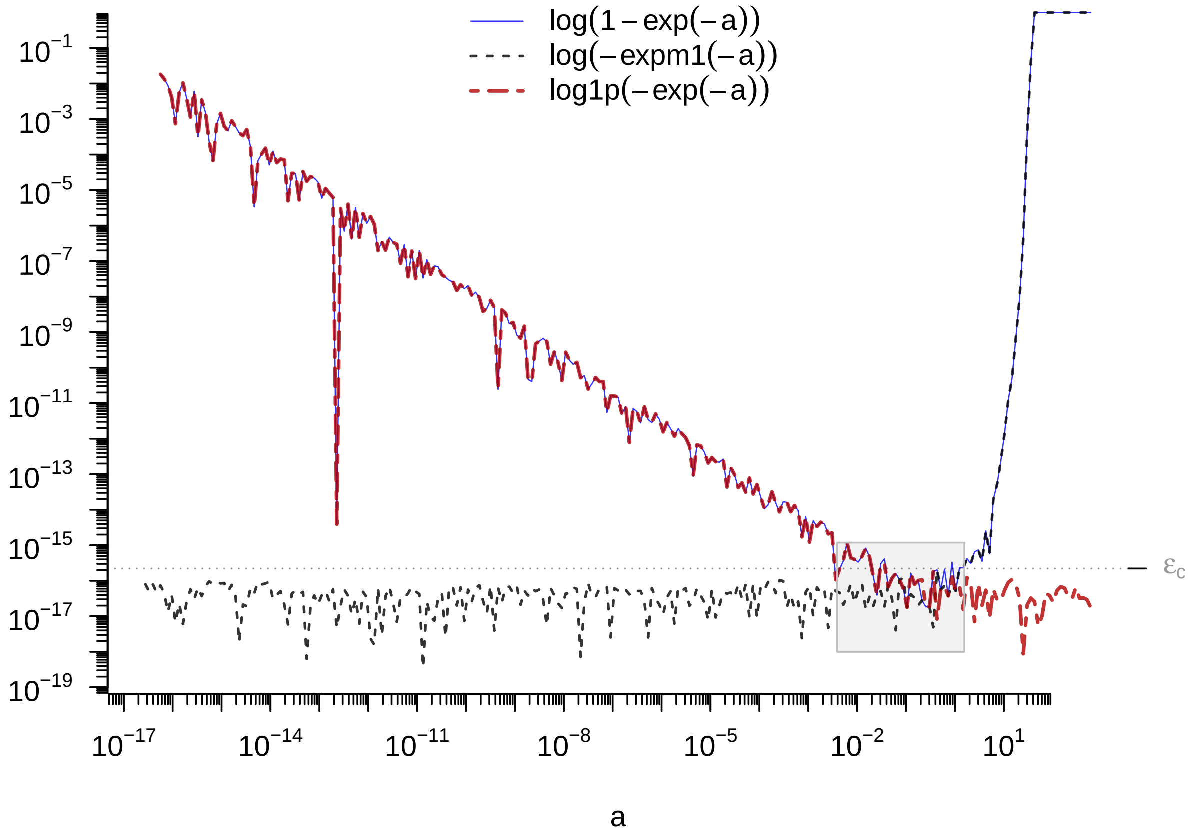

Figure 1: Absolute relative errors (with respect to 1024 bit **Rmpfr** computation) of the default, $\log(1-e^{-a})$, and the two methods "`expm1`" $\log(-\text{expm1}(-a))$ and "`log1p`" $\text{log1p}(-\exp(-a))$. Note that the default basically gives the maximum of the two methods' errors, whereas the final `log1mexp()` function will have the minimum (approximately) of the two. Figure 2 will be a zoom into the gray rectangular region where all three curves are close.

Figure 1 visualizes the relative errors of the three methods, where as in Figure 2, we zoom into the region where all methods have about the same (good) accuracy. The region is the rectangle defined by the ranges of `a.` and `ra2`:

```
> a. <- (1:400)/256
> ra <- t(sapply(a., relE.l1e))
> ra2 <- ra[,-1]
```

In addition, we want to smooth the two curves, using a method assuming approximately normal errors. Notice however that neither the original, nor the log-transformed values have approximately symmetric errors, so we use `MASS::boxcox()` to determine the "correct" power transformation,

```
> da <- cbind(a = a., as.data.frame(ra2))
> library(MASS)
> bc1 <- boxcox(abs(expm1) ~ a, data = da, lambda = seq(0,1, by=.01), plotit=.plot.BC)
> bc2 <- boxcox(abs(log1p) ~ a, data = da, lambda = seq(0,1, by=.01), plotit=.plot.BC)
> c(with(bc1, x[which.max(y)]),
    with(bc2, x[which.max(y)]))## optimal powers

[1] 0.38 0.30

> ## ==> taking ^ (1/3) :
> s1 <- with(da, smooth.spline(a, abs(expm1)^(1/3), df = 9))
> s2 <- with(da, smooth.spline(a, abs(log1p)^(1/3), df = 9))
```

and now plot a "zoom–in" of Figure 1. This already suggests that the cutoff, $a_0 = \log 2$ is empirically very close to optimal.

```
> matplot(a., abs(ra2), type = "l", log = "y", # ylim = c(-1,1)*1e-12,
          col=cc[-1], lwd=ll[-1], lty=lt[-1],
          ylim = yl, xlab = "a", ylab = "", axes=FALSE)
> legend("topright", leg[-1], col=cc[-1], lwd=ll[-1], lty=lt[-1], bty="n")
> eaxis(1); eaxis(2); draw.machEps()
> lines(a., predict(s1)$y ^ 3, col=cc[2], lwd=2)
> lines(a., predict(s2)$y ^ 3, col=cc[3], lwd=2)
```

Why is it very plausible to take $a_0 := \log 2$ as approximately optimal cutoff? Already from Figure 2, empirically, an optimal cutoff a_0 is around 0.7. We propose to compute

$$f(a) = \log\left(1 - e^{-a}\right) = \log(1 - \exp(-a)), \quad a > 0, \tag{5}$$

by a new method or function `log1mexp(a)`. It needs a cutoff a_0 between choosing `expm1` for $0 < a \le a_0$ and `log1p` for $a > a_0$, i.e.,

$$f(a) = \mathrm{log1mexp}(a) := \begin{cases} \log(-\mathrm{expm1}(-a)) & 0 < a \le a_0 \quad (:= \log 2 \approx 0.693) \\ \mathrm{log1p}(-\exp(-a)) & \qquad a > a_0. \end{cases} \tag{6}$$

The mathematical argument for choosing a_0 is quite simple, at least informally: In which situations does $1 - e^{-a}$ loose bits (binary digits) *entirely independently* of the computational algorithm? Well, as soon as it "spends" bits just to store its closeness to 1. And that is as soon as $e^{-a} < \frac{1}{2} = 2^{-1}$, because then, at least one bit cancels. This however is equivalent to $-a < \log(2^{-1}) = -\log(2)$ or $a > \log 2 =: a_0$.

3. Conclusion

We have used high precision arithmetic (R package **Rmpfr**) to empirically verify that computing $f(a) = \log\left(1 - e^{-a}\right)$ is accomplished best via equation (6).

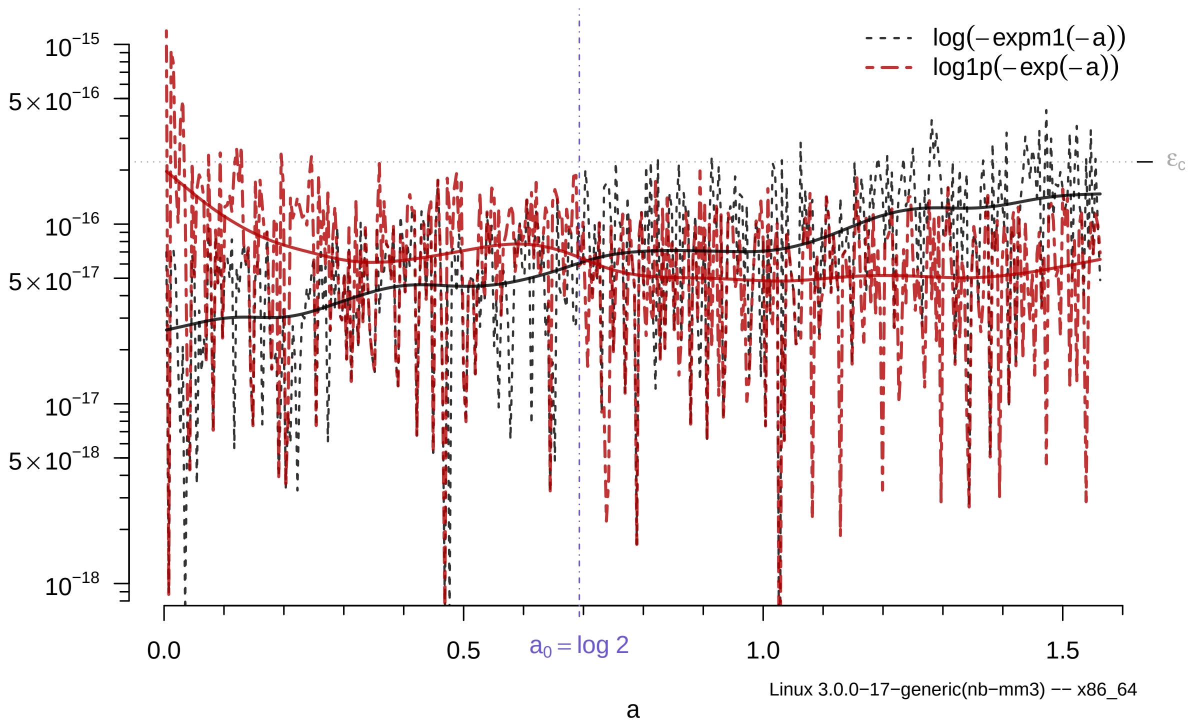

Figure 2: A "zoom in" of Figure 1 showing the region where the two basic methods, "`expm1`" and "`log1p`" switch their optimality with respect to their relative errors. Both have small relative errors in this region, typically below $\varepsilon_c :=$ `.Machine$double.eps` $= 2^{-52} \approx 2.22 \cdot 10^{-16}$. The smoothed curves indicate crossover close to $a = a_0 := \log 2$.

Session Information

```
> toLatex(sessionInfo())
```

- R version 2.15.0 Patched (2012-04-25 r59188), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=de_CH.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=de_CH.UTF-8`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=C`, `LC_PAPER=C`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=de_CH.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, graphics, grDevices, methods, stats, tools, utils
- Other packages: gmp 0.5-1, MASS 7.3-17, Rmpfr 0.4-6, sfsmisc 1.0-20
- Loaded via a namespace (and not attached): compiler 2.15.0

References

DiDonato AR, Morris Jr AH (1992). "Algorithm 708: Significant digit computation of the incomplete beta function ratios." *ACM Transactions on Mathematical Software*, **18**(3), 360–373. ISSN 0098-3500. URL `http://doi.acm.org/10.1145/131766.131776`.

IEEE, Open Group (2004). "The Open Group Base Specifications Issue 6 — log1p." In *IEEE Std 1003.1, 2004 Edition*. URL `http://pubs.opengroup.org/onlinepubs/009604599/functions/log1p.html`.

R Development Core Team (2004). *R: A language and environment for statistical computing (Ver. 1.9.0)*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3, URL `http://www.R-project.org`.

Affiliation:

Martin Mächler
Seminar für Statistik, HG G 16
ETH Zurich
8092 Zurich, Switzerland
E-mail: `maechler@stat.math.ethz.ch`
URL: `http://stat.ethz.ch/people/maechler`